Лекция 1

Полезные стандартные библиотеки Python

Python имеет ряд встроенных библиотек или наборов модулей, которые обеспечивают широкий спектр функций. Некоторые из наиболее часто используемых встроенных библиотек включают в себя:

- os: обеспечивает способ взаимодействия с операционной системой
- sys: обеспечивает доступ к системным параметрам и функциям
- math: предоставляет математические функции и константы
- random: генерирует случайные числа
- string: предоставляет функции, связанные со строками.
- re: обеспечивает операции сопоставления регулярных выражений
- json: обеспечивает кодирование и декодирование данных JSON.
- datetime: предоставляет классы для управления датами и временем.
- и друние.

Рассмотрим стандартную библиотеку Python, которая представляет собой набор модулей, включаемых в каждую установку Python. Эти модули обеспечивают широкий спектр функциональных возможностей и предназначены для повышения эффективности и простоты использования. С первыми модулями вы познакомились на занятиях по Python, поэтому сейчас мы рассмотрим другие модули.

collections: этот модуль предоставляет альтернативы встроенным типам, которые могут быть более эффективными в определенных ситуациях. Некоторые примеры классов в этой библиотеке включают Counter, который является подклассом dict для подсчета хешируемых объектов, и defaultdict, который является подклассом dict, который вызывает фабричную функцию для предоставления отсутствующих значений.

Counter: этот класс является подклассом dict для подсчета хешируемых объектов. Это контейнер, который будет содержать количество каждого из элементов, присутствующих в контейнере.

Пример использования:

```
from collections import Counter
c = Counter()
for word in ['red', 'blue', 'red', 'green', 'blue', 'blue']:
    c[word] += 1
print(c)
# Output: Counter({'blue': 3, 'red': 2, 'green': 1})
```

defaultdict: этот класс является подклассом dict, который вызывает фабричную функцию для предоставления отсутствующих значений. Он похож на стандартный словарь, но имеет значение по умолчанию для любых новых ключей, к которым осуществляется доступ.

Пример использования:

```
from collections import defaultdict
d = defaultdict(int)
d['a'] = 1
d['b'] = 2
print(d['c']) # 0
```

itertools: этот модуль предоставляет множество функций для работы с итераторами, включая функции для создания итераторов, их объединения и работы с бесконечными итераторами. Например, itertools.count() возвращает итератор, который создает бесконечную последовательность целых чисел, начиная с заданного значения, и itertools.product()возвращает декартово произведение входных итерируемых объектов в качестве итератора.

cycle: эта функция из itertools возвращает итератор, который будет бесконечно циклически перебирать элементы итерируемого объекта.

Пример использования:

```
from itertools import cycle
colors = cycle(['red', 'blue', 'green'])
for i in range(7):
    print(next(colors))
# Output: red blue green red blue green red
```

repeat: эта функция из itertools возвращает итератор, который будет повторять один и тот же элемент заданное количество раз или бесконечно, если число не указано.

Пример использования:

```
from itertools import repeat
for i in repeat('over', 5):
    print(i)
# Output: over over over over over
```

re: гемодуль обеспечивает операции сопоставления регулярных выражений. Регулярные выражения — это мощный инструмент для сопоставления шаблонов в тексте, а гемодуль предоставляет широкий набор функций для работы с регулярными выражениями в Python. Некоторые общие функции в этом модуле включают search(), findall(), sub() и compile().

Пример использования:

```
import re

# Search for the first occurrence of a pattern
result = re.search(r'\d{3}-\d{2}-\d{4}', 'Social Security Number: 123-45-6789')
print(result.group(0))  # 123-45-6789

# Find all occurrences of a pattern
result = re.findall(r'\b\w+\b', 'This is a sentence with a few words')
print(result)  # ['This', 'is', 'a', 'sentence', 'with', 'a', 'few', 'words']

# Replace all occurrences of a pattern with a new string
result = re.sub(r'\d{3}-\d{2}-\d{4}', 'xxx-xx-xxxx', 'Social Security Number: 123-45-6789')
print(result)  # Social Security Number: xxx-xx-xxxx
```

json: модуль обеспечивает кодирование и декодирование данных JSON (JavaScript Object Notation). JSON — это облегченный формат обмена данными, который легко читать и писать людям, а машинам легко анализировать и генерировать. Модуль jsonпредоставляет функции для работы с данными JSON в Python, включая load(), loads(), dump()и dumps().

Пример использования:

```
import json

# Serialize a Python object to a JSON formatted string
data = {'name': 'John', 'age': 30, 'city': 'New York'}
json_data = json.dumps(data)
print(json_data)  # {"name": "John", "age": 30, "city": "New York"}

# Deserialize a JSON string to a Python object
json_data = '{"name": "John", "age": 30, "city": "New York"}'
data = json.loads(json_data)
print(data)  # {'name': 'John', 'age': 30, 'city': 'New York'}
```

string: модуль предоставляет функции, связанные со строками. Он содержит ряд полезных строковых констант, таких как ascii_letters, digitsи punctuation, а также функции для работы со строками, такие как capwords(), join(), replace()и split().

Пример использования:

```
import string

# Capitalize the first letter of each word in a string
result = string.capwords('this is a test')
```

```
print(result)  # This Is A Test

# Concatenate a list of strings with a separator
words = ['hello', 'world']
```

Все эти библиотеки очень полезны и могут использоваться в различных ситуациях. Они предоставляют функциональные возможности, которые часто необходимы в программировании, и они хорошо протестированы и оптимизированы для обеспечения эффективности.

JSON и CSV

JSON (обозначение объектов JavaScript) и CSV (значения, разделенные запятыми) — два широко используемых формата данных для обмена и хранения данных.

JSON — это удобочитаемый текстовый формат, который обычно используется для обмена данными в API и веб-приложениях. Он использует пары ключ-значение для представления данных и часто используется для хранения сложных структур данных, таких как вложенные массивы и словари. JSON также легко анализировать и генерировать с использованием многих языков программирования, включая Python.

CSV, с другой стороны, является более простым и широко поддерживаемым форматом. Он хранит данные в табличной форме, где каждая строка представляет собой запись, а каждое поле в записи отделяется запятой. Этот формат широко используется в приложениях для анализа данных и электронных таблиц и поддерживается большинством программ баз данных и электронных таблиц. Простота CSV упрощает использование и передачу данных, но не подходит для представления сложных структур данных.

Таким образом, JSON является более универсальным и гибким форматом данных, в то время как CSV проще и шире поддерживается. Выбор между JSON и CSV зависит от конкретных потребностей хранимых данных и приложений, которые будут к ним обращаться.

Когда использовать JSON и CSV:

- **JSON: если вы хотите обмениваться данными между веб-приложением и сервером,** то лучше использовать JSON. Например, вы можете использовать API для извлечения данных с сервера и получения данных в формате JSON. Затем вы можете использовать данные JSON в своем веб-приложении.
- **CSV: если вы хотите проанализировать данные в программе для работы с электронными таблицами**, лучше использовать CSV. Например, вы можете экспортировать данные из базы данных и сохранить их в виде файла CSV, который затем можно открыть в программе для работы с электронными таблицами, такой как Microsoft Excel или Google Sheets. Вы можете легко

сортировать, фильтровать и визуализировать данные в программе для работы с электронными таблицами.

- **JSON: если вы хотите хранить сложные структуры данных**, то лучше использовать JSON. Например, вы можете использовать JSON для хранения информации о коллекции книг, где у каждой книги есть название, автор, дата публикации и список жанров. Вы можете легко вложить эти данные в формат JSON, чтобы представить отношения между различными элементами.
- **CSV: если вы хотите хранить данные, которые легко обрабатывать**, скорее всего более удобным будет использовать CSV. Например, вы можете использовать CSV для хранения списка имен и адресов, где каждая строка представляет собой запись, а каждое поле в записи отделяется запятой. Вы можете легко манипулировать этими данными, используя простые строковые операции, и записывать их в файл для последующего использования.

Примеры кода:

Пример с JSON:

```
import json

# A Python dictionary to be converted to JSON
person = {
  "first_name": "John",
  "last_name": "Doe",
  "age": 30,
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA"
  }
}

# Convert the dictionary to JSON string
person_json = json.dumps(person)

# Load the JSON string to a Python dictionary
person_dict = json.loads(person_json)

print("Person JSON:", person_json)
print("Person Dict:", person_dict)
```

Пример с CSV:

```
import csv

# A list of dictionaries to be converted to CSV
```

```
people = [
  {
    "first_name": "John",
    "last_name": "Doe",
    "age": 30,
    "address": "123 Main St, Anytown, CA"
  },
  {
    "first_name": "Jane",
    "last_name": "Doe",
    "age": 28,
    "address": "456 Elm St, Anytown, CA"
  }
]

# Write the list of dictionaries to a CSV file
with open('people.csv', 'w', newline='') as csvfile:
    fieldnames = ['first_name', 'last_name', 'age', 'address']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    for person in people:
        writer.writerow(person)

# Read the CSV file to a list of dictionaries
with open('people.csv', 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    people_list = list(reader)

print("People List:", people_list)
```

В приведенных выше примерах JSON используется для представления одного словаря, а CSV — для представления списка словарей. Преобразование между двумя форматами можно выполнить с помощью библиотек json и csv.

Практические задания по теме использования модулей re, json, stringи csv в Python:

Обработка данных JSON: с помощью json модуля напишите скрипт, который считывает файл JSON, содержащий информацию о книгах (название, автор и ISBN), и выводит список всех книг, написанных конкретным автором.

Пример

```
import json

def find_books_by_author(file_path, author):
    with open(file_path, 'r') as f:
        data = json.load(f)
    books = []
    for book in data['books']:
        if book['author'] == author:
            books.append(book)
    return books

file_path = 'books.json'
author = 'J.K. Rowling'
books = find_books_by_author(file_path, author)
print(books)
```

Очистка текстовых данных. Используя модуль string, напишите сценарий, который принимает текстовый файл и удаляет все знаки препинания и заглавные буквы, а затем записывает очищенный текст в новый файл.

Пример

```
import string

def clean_text(file_path):
    with open(file_path, 'r') as f:
        text = f.read()
    text = text.lower()
    text = text.translate(text.maketrans('', '', string.punctuation))
    with open('cleaned_text.txt', 'w') as f:
        f.write(text)

file_path = 'text.txt'
clean_text(file_path)
```

Манипулирование данными CSV: Используя модуль csv, напишите скрипт, который читает CSV-файл, выполняет вычисления с данными и выводит результаты в новый CSV-файл.

Пример

```
import csv

def calculate_avg_income(file_path):
    with open(file_path, 'r') as f:
        reader = csv.DictReader(f)
```

```
    income_by_state = {}
    for row in reader:
        state = row['state']
        income = int(row['income'])
        if state in income_by_state:
            income_by_state[state]['total'] += income
            income_by_state[state]['count'] += 1
        else:
            income_by_state[state] = {'total': income, 'count': 1}
                avg_income_by_state = {state: income_by_state[state]['total'] / income_by_state[state]['count'] for state in income_by_state}
    return avg_income_by_state

file_path = 'income_data.csv'
avg_income = calculate_avg_income(file_path)
print(avg_income)
```

Подсчет вхождений элементов в список. Используя класс Counter, напишите сценарий, который принимает список элементов и возвращает словарь элементов и их вхождений в списке.

Пример

```
from collections import Counter

def count_items(items):
    return Counter(items)

items = [1, 2, 3, 4, 1, 2, 3, 1, 2, 1]
counts = count_items(items)
print(counts)
```

Доступ к отсутствующим ключам словаря. Используя класс defaultdict, напишите сценарий, который принимает список элементов и значение по умолчанию и возвращает словарь элементов и их вхождений в списке. Если элемента нет в словаре, он должен вернуть значение по умолчанию.

Пример

```
from collections import defaultdict

def count_items(items, default_value):
    counts = defaultdict(lambda: default_value)
    for item in items:
```

```
        counts[item] += 1
    return counts

items = [1, 2, 3, 4, 1, 2, 3, 1, 2, 1]
default_value = 0
counts = count_items(items, default_value)
print(counts)
```

Перебор комбинаций: используя модуль itertools, напишите скрипт, который принимает список элементов и число nи возвращает все возможные комбинации nэлементов из списка.

Пример

```
import itertools

def get_combinations(items, n):
    return list(itertools.combinations(items, n))

items = [1, 2, 3, 4]
n = 2
combinations = get_combinations(items, n)
print(combinations)
```

Повторение итератора. Используя модуль itertools, напишите сценарий, который принимает итератор и число n, и возвращает повторяющийся итератор n.

Пример

```
import itertools

def repeat_iterator(iterable, n):
    return itertools.chain.from_iterable(itertools.repeat(iterable, n))

iterable = [1, 2, 3]
n = 2
repeated_iterable = repeat_iterator(iterable, n)
print(list(repeated_iterable))
```

Циклическое перебор итератора. Используя модуль itertools, напишите скрипт, который принимает итератор и возвращает бесконечный итератор, циклически перебирающий входной итератор.

Пример

```
import itertools

def cycle_iterator(iterable):
    return itertools.cycle(iterable)

iterable = [1, 2, 3]
cycled_iterable = cycle_iterator(iterable)
print(next(cycled_iterable))
print(next(cycled_iterable))
print(next(cycled_iterable))
print(next(cycled_iterable))
```

Лекция 2

Эффективные структуры данных

pandas — это библиотека для обработки и анализа данных. Она предоставляет структуры данных и функции для обработки и манипулирования числовыми таблицами и данными временных рядов. Она построена на основе numpy и предоставляет простые в использовании структуры данных и инструменты анализа данных для обработки и манипулирования числовыми таблицами и данными временных рядов.

numpy — это библиотека для языка программирования Python, добавляющая поддержку больших многомерных массивов и матриц, а также большой набор высокоуровневых математических функций для работы с этими массивами.

pandas является очень мощной библиотекой для обработки и анализа данных и может выполнять широкий спектр задач.

Группировка и агрегирование: вы можете группировать данные по одному или нескольким столбцам и применять к каждой группе различные функции агрегирования. Например, вы можете сгруппировать набор данных по значениям в столбце, а затем вычислить среднее значение для каждой группы.

Пример
```
import pandas as pd
df = pd.read_csv('data.csv')
grouped_data = df.groupby('column_name').mean()
print(grouped_data)
```

Обработка отсутствующих данных: pandas предоставляет различные варианты обработки отсутствующих данных, такие как заполнение отсутствующих значений определенным значением или интерполяция отсутствующих значений.

Пример
```
import pandas as pd
df = pd.read_csv('data.csv')
df = df.fillna(value=0)
print(df)
```

Слияние и объединение данных: pandas предоставляет различные варианты слияния и объединения данных из нескольких фреймов данных или наборов данных. Например, вы можете соединить два DataFrames в определенном столбце, подобно SQL JOIN.

Пример
```
import pandas as pd
df1 = pd.read_csv('data1.csv')
```

```
df2 = pd.read_csv('data2.csv')
merged_data = pd.merge(df1, df2, on='column_name')
print(merged_data)
```

Преобразование данных: pandas предоставляет различные функции для преобразования данных. Например, вы можете повернуть DataFrame, чтобы преобразовать его в другой формат.

Пример

```
import pandas as pd
df = pd.read_csv('data.csv')
pivoted_data = df.pivot(index='column_name1', columns='column_name2', values='column_name3')
print(pivoted_data)
```

pivot - это метод DataFrame в pandas, который можно использовать для изменения формы DataFrame. Это позволяет вам преобразовать DataFrame из «длинного» формата в «широкий» формат или наоборот.

Основной синтаксис для использования метода pivot следующий:

```
pivot_table = df.pivot(index='index_column', columns='column_to_become_columns', values='column_to_become_values')
```

Здесь index_columnстолбец, который станет индексом нового кадра данных, column_to_become_columnsстолбец, который станет столбцами нового кадра данных, и column_to_become_valuesстолбец, который будет заполнять ячейки нового кадра данных.

Например, предположим, что у вас есть DataFrame с тремя столбцами: «дата», «продукт» и «продажи». Чтобы повернуть этот DataFrame так, чтобы столбец «дата» стал индексом, столбец «продукт» стал столбцами, а столбец «продажи» стал значениями, вы должны использовать следующий код:

Пример

```
import pandas as pd
df = pd.read_csv('data.csv')
pivoted_data = df.pivot(index='date', columns='product', values='sales')
print(pivoted_data)
```

Еще одна полезная функция pivot — возможность передать aggfunc ей необязательный аргумент. Это позволяет указать функцию агрегирования, которая применяется к повторяющимся значениям. По умолчанию pivot вызывает ошибку, если есть повторяющиеся значения.

Например, если у вас есть следующий DataFrame:

Пример

```
  Product  Sales  Date
0    Apple    100    1
1    Apple    200    1
2     Pear     50    2
3     Pear     30    2
```

Вы можете использовать следующий код для поворота этого фрейма данных и суммирования повторяющихся значений:

Пример

```
import pandas as pd
df = pd.read_csv('data.csv')
pivoted_data = df.pivot_table(index='Date', columns='Product', values='Sales',aggfunc='sum')
print(pivoted_data)
```

В результате сводная таблица будет иметь вид:

Пример

```
Product  Apple  Pear
Date
1            300   NaN
2             NaN   80
```

Обратите внимание, что pivot_table похоже на повторяющиеся значения pivot, но будет обрабатывать повторяющиеся значения, применяя указанную функцию агрегирования.

Визуализация данных: pandas хорошо интегрируется с библиотеками визуализации данных, такими как matplotlib и seaborn, что позволяет легко создавать графики и диаграммы из ваших фреймов данных.

Пример

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot(kind='bar', x='column_name1',y='column_name2')
plt.show()
```

matplotlib - это библиотека построения графиков для языка программирования Python и его расширения для числовой математики NumPy. Она предоставляет объектно-ориентированный API для встраивания графиков в приложения с помощью инструментов общего назначения с графическим интерфейсом, таких как Tkinter, wxPython, Qt или GTK.

Она предоставляет ряд инструментов для создания статических, анимированных и интерактивных визуализаций в Python. Некоторые из наиболее распространенных типов графиков, которые можно создать, matplotlib включают:

- Линейные графики: для отображения данных, которые изменяются во времени или других непрерывных переменных.
- Точечные диаграммы: для отображения взаимосвязи между двумя или более переменными.
- Гистограммы: для отображения данных, разделенных на разные категории.
- Гистограммы: для отображения распределения одной переменной.
- Круговые диаграммы: для отображения доли различных категорий в наборе данных.

Вот пример того, как использовать matplotlib для создания простого линейного графика:

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a figure and axis
fig, ax = plt.subplots()

# Plot the data
ax.plot(x, y)

# Add labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Line Plot')

# Show the plot
plt.show()
```

matplotlib также имеет широкие возможности настройки, вы можете легко изменить цвет, стиль линий и стиль маркеров ваших графиков. Вы также можете добавлять линии сетки, легенды и аннотации к своим графикам. Кроме того, вы можете

использовать различные предустановленные стили и цветовые палитры, такие как «ggplot» или «seaborn».

Есть и другие библиотеки, такие как seaborn, которые построены поверх matplotlib и предоставляют высокоуровневый интерфейс для рисования привлекательных и информативных статистических графиков.

Для более продвинутого использования matplotlib также есть возможности для создания 3D-графиков, подграфиков и других расширенных визуализаций. Проявив немного творчества и зная возможности библиотеки, вы сможете создавать всевозможные интересные и информативные визуализации с помощью matplotlib.

Визуализация данных из файла CSV:

Пример

```
import matplotlib.pyplot as plt
import pandas as pd

# Read data from a CSV file
data = pd.read_csv('data.csv')

# Create a scatter plot of the data
plt.scatter(data['x'], data['y'])
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot of Data')
plt.show()
```

Визуализация распределения набора данных:

Пример

```
import matplotlib.pyplot as plt
import numpy as np

# Create a random dataset
data = np.random.normal(50, 10, 100)

# Create a histogram of the data
plt.hist(data, bins=20)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Data')
```

```
plt.show()
```
Сравнение нескольких наборов данных на одном графике:

Пример
```
import matplotlib.pyplot as plt
import numpy as np

# Create two random datasets
data1 = np.random.normal(50, 10, 100)
data2 = np.random.normal(60, 15, 100)

# Create a figure and axis
fig, ax = plt.subplots()

# Plot both datasets on the same axis
ax.plot(data1, label='Dataset 1')
ax.plot(data2, label='Dataset 2')

# Add labels, title, and legend
ax.set_xlabel('Index')
ax.set_ylabel('Value')
ax.set_title('Comparison of Two Datasets')
ax.legend()

# Show the plot
plt.show()
```

Построение математической функции:

Пример
```
import matplotlib.pyplot as plt
import numpy as np

# Create x and y data
x = np.linspace(-np.pi, np.pi, 100)
y = np.sin(x)

# Create a figure and axis
fig, ax = plt.subplots()

# Plot the data
ax.plot(x, y)

# Add labels and title
```

```
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Plot of the Sine Function')

# Show the plot
plt.show()
```

Моделирование простой анимации:

Пример

```
import matplotlib.pyplot as plt
import numpy as np

# Create x and y data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# Create a figure and axis
fig, ax = plt.subplots()

# Initialize a line object
line, = ax.plot(x, y)

# Define an animation function
def animate(i):
    line.set_ydata(np.sin(x + i / 10))
    return line,

# Create an animation object
ani = animation.FuncAnimation(fig, animate, frames=100, blit=True)

# Show the animation
plt.show()
```

Это всего лишь несколько примеров множества типов графиков и визуализаций, которые вы можете создавать с помощью matplotlib. Библиотека предоставляет широкий спектр параметров настройки, упрощая создание уникальных и информативных визуализаций для ваших данных.

Дополнительную информацию и примеры можно найти на официальном сайте matplotlib: https://matplotlib.org/stable/contents.html.

Практические задачи и примеры использования pandas и NumPy

Загрузка CSV-файла в DataFrame: Используя pandas, напишите скрипт, который загружает CSV-файл в DataFrame и отображает первые 5 строк DataFrame.

Пример
```
import pandas as pd

df = pd.read_csv('data.csv')
print(df.head())
```

Выбор столбцов из DataFrame: Используя pandas, напишите сценарий, который выбирает определенные столбцы из DataFrame и отображает их.

Пример
```
import pandas as pd

df = pd.read_csv('data.csv')
columns_to_select = ['col1', 'col2', 'col3']
selected_columns = df[columns_to_select]
print(selected_columns)
```
Фильтрация строк из DataFrame : Используя pandas, напишите скрипт, который фильтрует строки из DataFrame на основе определенного условия и отображает полученный DataFrame.

Пример
```
import pandas as pd

df = pd.read_csv('data.csv')
condition = df['col1'] > 5
filtered_df = df[condition]
print(filtered_df)
```

Вычисление статистики для массива numpy. Используя numpy, напишите скрипт, который загружает файл CSV в массив numpy и вычисляет среднее значение, стандартное отклонение и максимальное значение массива.

Пример
```
import numpy as np

data = np.genfromtxt('data.csv', delimiter=',')
mean = np.mean(data)
std = np.std(data)
```

```
max_value = np.max(data)
print("Mean:", mean)
print("Standard deviation:", std)
print("Max value:", max_value)
```

Операции с матрицами: Используя numpy, напишите сценарий, который создает матрицу и выполняет основные математические операции, такие как сложение, вычитание, умножение и транспонирование матрицы.

Пример

```
import numpy as np

matrix1 = np.array([[1,2,3],[4,5,6]])
matrix2 = np.array([[7,8,9],[10,11,12]])

#Addition
add_matrix = matrix1 + matrix2
print("Addition of matrices:", add_matrix)

#Subtraction
sub_matrix = matrix1 - matrix2
print("Subtraction of matrices:", sub_matrix)

#multiplication
mul_matrix = matrix1 * matrix2
print("Multiplication of matrices:", mul_matrix)
```

Лекция 3

Работа с объектами MS Office

Модуль win32com на Python позволяет взаимодействовать с приложениями Microsoft Office, такими как Word, Excel и PowerPoint, с помощью интерфейса Component Object Model (COM).

Например, вы можете использовать его, чтобы открыть существующий документ Word, добавить текст, изменить шрифт и сохранить документ:

```
import win32com.client as win32

word = win32.gencache.EnsureDispatch('Word.Application')
doc = word.Documents.Open('C:\\path\\to\\document.docx')
word.Visible = True

range = doc.Range()
range.InsertAfter("Hello, world!")
range.Font.Name = "Arial"
range.Font.Size = 14

doc.Save()
doc.Close()
word.Quit()
```

В Excel вы можете открыть существующую книгу, добавить данные на лист, создать диаграммы и графики и сохранить книгу:

```
import win32com.client as win32

excel = win32.gencache.EnsureDispatch('Excel.Application')
workbook = excel.Workbooks.Open('C:\\path\\to\\workbook.xlsx')
excel.Visible = True

worksheet = workbook.Worksheets("Sheet1")
worksheet.Cells(1, 1).Value = "Name"
worksheet.Cells(1, 2).Value = "Age"
worksheet.Cells(2, 1).Value = "John"
worksheet.Cells(2, 2).Value = 30

chart = worksheet.ChartObjects().Add(0,0,300,300).Chart
chart.ChartType = win32.constants.xlColumnClustered
chart.SetSourceData(worksheet.Range("A1:B2"))

workbook.Save()
workbook.Close()
```

```
excel.Quit()
```

Вы также можете использовать win32com для взаимодействия с PowerPoint, например, вы можете создать новую презентацию, добавить слайды, добавить текст и изображения и сохранить презентацию:

```
import win32com.client as win32

ppt = win32.gencache.EnsureDispatch('PowerPoint.Application')
presentation = ppt.Presentations.Add()
ppt.Visible = True

slide = presentation.Slides.Add(1, win32.constants.ppLayoutTitleOnly)
title = slide.Shapes.Title
title.TextFrame.TextRange.Text = "My Presentation"

slide = presentation.Slides.Add(2, win32.constants.ppLayoutBlank)
textbox = slide.Shapes.AddTextbox(1,20,200,100,50)
textbox.TextFrame.TextRange.Text = "Hello, World!"

image = slide.Shapes.AddPicture("C:\\path\\to\\image.jpg", False, True, 20, 20, 100, 100)

presentation.SaveAs("C:\\path\\to\\presentation.pptx")
presentation.Close()
ppt.Quit()
```

Работать с файлами .docx и .xlsx в Python можно с помощью библиотек docx и openpyxl соответственно.

docx позволяет читать и редактировать существующие файлы .docx, а также создавать новые файлы .docx с нуля. В файл .docx можно добавлять текст, абзацы, списки, таблицы и изображения.

Например, вы можете использовать следующий код, чтобы создать новый файл .docx и добавить к нему заголовок и некоторый текст:

```
from docx import Document

document = Document()
document.add_heading('My Heading', 0)
document.add_paragraph('My paragraph')
document.save('My document.docx')
```

openpyxl позволяет читать и записывать файлы .xlsx. Вы можете получить доступ к ячейкам, строкам и столбцам электронной таблицы и изменить их значения. Вы также

можете создавать новые электронные таблицы и добавлять в них диаграммы, формулы и изображения.

Например, вы можете использовать следующий код для чтения данных из существующего файла .xlsx и печати значения ячейки в первой строке и первом столбце:

```
from openpyxl import load_workbook

workbook = load_workbook('data.xlsx')
worksheet = workbook.active
cell_value = worksheet.cell(row=1, column=1).value
print(cell_value)
```

Aspose.Words — это библиотека .NET, которая позволяет работать с документами Microsoft Word. Он предлагает широкий спектр функций и преимуществ, включая возможность работы со стилями в документах Word.

Одной из ключевых особенностей Aspose.Words является возможность создавать, изменять и форматировать стили в документе Word. Вы можете создавать собственные стили или изменять существующие в соответствии с вашими потребностями. Кроме того, вы можете применять стили к тексту, абзацам или таблицам в документе для обеспечения единообразия и упрощения форматирования.

Еще одним преимуществом использования Aspose.Words является возможность программно выполнять операции над документами Word. Например, вы можете вставлять или удалять текст, добавлять или изменять изображения, а также обновлять таблицы и списки. Вы также можете выполнять операции с элементами документа, такими как абзацы, таблицы и разделы.

Одной из ключевых особенностей Aspose.Words является возможность работы со стилями в документах Word. С Aspose.Words вы можете легко получить доступ и изменить стили в документе, включая шрифт, размер, цвет и выравнивание. Вы также можете создавать новые стили и применять их к определенным частям текста в документе.

Например, чтобы изменить шрифт всех экземпляров стиля «Заголовок 1» в документе, вы можете использовать следующий код:

```
// Load the document
Document doc = new Document("input.docx");

// Access the "Heading 1" style
Style heading1 = doc.getStyles().getByName("Heading 1");
```

```
// Change the font of the style
heading1.getFont().setName("Arial");

// Save the updated document
doc.save("output.docx");
```

В этом примере Documentкласс используется для загрузки входного документа, а getStylesметод используется для доступа к стилям, определенным в документе. Затем getByNameметод используется для доступа к стилю «Заголовок 1», а getFontметод используется для доступа к шрифту, связанному со стилем. Наконец, setNameметод используется для изменения имени шрифта, а saveметод используется для сохранения обновленного документа.

В целом, Aspose.Words предоставляет полный набор функций для работы с документами и стилями Word, что делает его идеальным решением для разработчиков, которым необходимо автоматизировать задачи обработки документов Word.

Объектно-ориентированное программирование (ООП) — это парадигма программирования, основанная на концепции объектов, обладающих свойствами (также известными как атрибуты) и методами. В Python классы используются для определения объектов, их свойств и методов.

Класс — это план создания объектов (определенной структуры данных), предоставления начальных значений для состояния (переменные-члены или атрибуты) и реализации поведения (функции-члены или методы).

Вот пример простого класса в Python:

```
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def bark(self):
        print("Woof!")

dog1 = Dog("Fido", "Golden Retriever")
dog1.bark() # Output: Woof!
```

В этом примере мы определяем класс Dogc двумя атрибутами nameи breed. Метод __init__— это специальный метод, который вызывается при создании нового объекта этого класса. Он инициализирует атрибуты объекта. В классе также есть метод bark, который печатает "Гав!" когда звонили.

Затем мы можем создать объект класса Dog, вызвав имя класса как функцию и передав необходимые аргументы для __init__метода.

В ООП классы наследуются от других классов и могут добавлять или переопределять свои методы и атрибуты. Это обеспечивает чистую и организованную структуру кода с меньшим количеством повторяющихся кодов. Вы также можете использовать наследование для создания иерархии классов, которая организует классы в соответствии с их функциональностью и отношениями.

Пример класса для банковского счета с методами пополнения, снятия и проверки баланса:

```
class BankAccount:
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"{amount} has been deposited to {self.name}'s account")

    def withdraw(self, amount):
        if amount > self.balance:
            print(f"Insufficient balance in {self.name}'s account")
        else:
            self.balance -= amount
            print(f"{amount} has been withdrawn from {self.name}'s account")

    def check_balance(self):
        print(f"{self.name}'s current balance is {self.balance}")

account1 = BankAccount("John Doe", 1000)
account1.check_balance() # Output: John Doe's current balance is 1000
account1.deposit(500) # Output: 500 has been deposited to John Doe's account
account1.check_balance() # Output: John Doe's current balance is 1500
account1.withdraw(2000) # Output: Insufficient balance in John Doe's account
account1.withdraw(500) # Output: 500 has been withdrawn from John Doe's account
account1.check_balance() # Output: John Doe's current balance is 1000
```

Этот класс имеет три метода: deposit, withdraw, и check_balance. Каждый метод выполняет определенное действие над объектом банковского счета. Метод depositпринимает сумму в качестве входных данных и добавляет ее к балансу счета. Метод withdraw принимает сумму в качестве входных данных, проверяет, есть ли на

счете достаточный баланс, и если да, то вычитает сумму из баланса. Метод check_balance просто печатает текущий баланс счета.

В этом примере мы создаем объект account1 класса BankAccountи вызываем методы объекта, чтобы продемонстрировать, как работает класс.

Другой пример — создание класса для прямоугольника с атрибутами длины и ширины и методами вычисления площади и периметра:

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)

rect1 = Rectangle(5, 10)
print(rect1.calculate_area()) # Output: 50
print(rect1.calculate_perimeter()) # Output: 30
```

В этом примере класс Rectangle имеет два атрибута: lengthи width. Он также имеет два метода, calculate_area и calculate_perimeter, которые возвращают площадь и периметр прямоугольника соответственно.

В объектно-ориентированном программировании (ООП) концепции инкапсуляции, наследования и полиморфизма играют фундаментальную роль в создании эффективного, поддерживаемого и масштабируемого кода.
Инкапсуляция относится к концепции упаковки данных и функций в единый блок или объект. Это позволяет скрывать информацию, делая код более безопасным и снижая риск ошибок.

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self._speed = 0

    def accelerate(self):
        self._speed += 10
```

```
    def brake(self):
        self._speed -= 10

    def get_speed(self):
        return self._speed
```

Наследование — механизм, с помощью которого один класс наследует атрибуты и поведение другого класса. Это позволяет создавать новые классы на основе существующих классов, уменьшая дублирование кода и способствуя повторному использованию кода.

```
def show_shape_area(shape):
    print("Area:", shape.area())

class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

rect = Rectangle(10, 20)
circle = Circle(5)

show_shape_area(rect)
show_shape_area(circle)
```

Полиморфизм позволяет создавать объекты, которые имеют одинаковый интерфейс, но могут вести себя по-разному. Это делает код более гибким, позволяя заменять объекты без нарушения кода.

```
def show_shape_area(shape):
    print("Area:", shape.area())

class Rectangle:
```

```
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

rect = Rectangle(10, 20)
circle = Circle(5)

show_shape_area(rect)
show_shape_area(circle)
```

В Python декораторы используются для изменения поведения функции или класса. **Декораторы** — функции, которые принимают другую функцию в качестве аргумента и возвращают новую функцию, которая обычно расширяет поведение исходной функции.

```
def uppercase(func):
    def wrapper():
        original_result = func()
        modified_result = original_result.upper()
        return modified_result
    return wrapper

@uppercase
def greet():
    return "Hello, World!"

print(greet())
```

Перегрузка оператора — способность объекта изменять поведение оператора в зависимости от типа операндов. Это позволяет создавать пользовательские объекты, которые могут вести себя естественным и интуитивно понятным образом, облегчая чтение и запись кода.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
```

```
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    def __str__(self):
        return f"Vector({self.x}, {self.y})"

v1 = Vector(1, 2)
v2 = Vector(3, 4)
v3 = v1 + v2
print(v3)  # Output: Vector(4, 6)
```

В этом примере мы определили класс Vector, представляющий двумерный вектор. Метод __add__используется для определения поведения +оператора при его применении к двум Vectorобъектам. Этот __str__метод используется для определения того, как Vectorобъект должен быть представлен в виде строки, что полезно для целей отладки и тестирования.

Магические методы, также известные как методы dunder, — это специальные методы Python, которые имеют префикс и суффикс с двойным подчеркиванием. Эти методы используются для реализации определенных специальных действий в Python, таких как перегрузка операторов и создание и уничтожение объектов.

```
class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages

    def __str__(self):
        return f"{self.title} by {self.author}, {self.pages} pages"

book = Book("The Great Gatsby", "F. Scott Fitzgerald", 180)
print(book)
```

Классы исключений используются для представления ошибок в коде. Эти классы позволяют создавать собственные сообщения об ошибках, упрощая выявление и исправление ошибок в коде.

```
class InvalidAgeException(Exception):
    pass

age = int(input("Enter your age: "))
if age < 0:
    raise InvalidAgeException("Age cannot be negative")
```

Классы контекстных менеджеров — это тип объекта, который можно использовать в операторе with. Они позволяют управлять ресурсами, такими как файлы или соединения с базой данных, простым и эффективным образом.

```
class FileManager:
    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode

    def __enter__(self):
        self.file = open(self.filename, self.mode)
        return self.file

    def __exit__(self, exc_type, exc_value, exc_tb):
        self.file.close()

with FileManager("example.txt", "w") as file:
    file.write("Hello, World!")
```

Enum — это встроенный тип Python, который можно использовать для определения набора именованных констант. Это делает код более читабельным и удобным для сопровождения, так как снижает риск ошибок из-за использования в коде жестко заданных значений.

```
from enum import Enum

class Color(Enum):
    RED = 1
    GREEN = 2
    BLUE = 3

print(Color.RED)
print(Color.RED.value)
```

Практические задачи по ООП:

- Создайте класс для банковского счета с методами пополнения, снятия и проверки баланса.
- Создайте класс для автомобиля с методами запуска, остановки и подачи сигнала.
- Создайте класс для человека с атрибутами для имени, возраста и адреса.
- Создайте класс для университета с атрибутами имени, адреса и студентов.
- Создайте класс для прямоугольника с атрибутами длины и ширины и методами вычисления площади и периметра.

- Создайте класс для корзины покупок с методами добавления и удаления товаров и расчета общей стоимости.
- Создайте класс для игры с атрибутами для очков, жизней и уровня, а также методы для запуска, завершения и перезапуска игры.

Дополнительную информацию и примеры вы можете найти на официальном сайте python: https://docs.python.org/3/tutorial/classes.html.

Лекция 4

Оптимизация, линейная алгебра и машинное обучение

Оптимизация — это процесс поиска наилучшего решения проблемы в рамках набора ограничений. С математической точки зрения, это включает в себя поиск значений набора переменных, которые минимизируют или максимизируют заданную функцию, называемую целевой функцией.

Оптимизация широко используется во многих областях, таких как финансы, инженерия и исследование операций. Существует множество различных методов оптимизации, таких как линейное программирование, нелинейное программирование, целочисленное программирование и стохастическая оптимизация, которые можно использовать для решения различных типов задач оптимизации.

В Python есть несколько библиотек, таких как scipy.optimize, которые предоставляют широкий спектр алгоритмов оптимизации, таких как градиентный спуск, Nelder-Mead, сопряженный градиент и BFGS. Эти библиотеки можно использовать для оптимизации различных типов функций, включая линейные и нелинейные функции, с ограничениями или без них.

SciPy — это библиотека, содержащая набор научных и численных алгоритмов, включая оптимизацию, линейную алгебру, интеграцию, интерполяцию и многое другое. Он построен на основе NumPy и предоставляет дополнительные функции для научных вычислений.

Например, вы можете использовать функцию минимизации из модуля scipy.optimize, чтобы минимизировать функцию.
Вы можете использовать следующий код для минимизации функции f(x) = x^2 с помощью алгоритма BFGS:

```
from scipy.optimize import minimize

def f(x):
    return x**2

res = minimize(f, x0=2, method='BFGS')
print(res.x) #The minimum point
```

Также можно оптимизировать функции с ограничениями, используя функцию минимизации, предоставив словарь ограничений или используя метод «SLSQP»:

```
from scipy.optimize import minimize

def f(x):
    return x[0]**2 + x[1]**2
```

```
def constraint1(x):
    return x[0] + x[1] - 1

def constraint2(x):
    return 1 - x[0] - x[1]

cons = [{'type': 'eq', 'fun': constraint1},
        {'type': 'ineq', 'fun': constraint2}]

res = minimize(f, x0=[0.5, 0.5], constraints=cons)
print(res.x) #The minimum point
```

Аппроксимация кривой — это процесс поиска наилучшей кривой, описывающей взаимосвязь между независимыми и зависимыми переменными. SciPy предоставляет curve_fitфункцию в scipy.optimizeмодуле, которую можно использовать для подгонки кривой к набору точек данных. Функция принимает функцию модели, начальные параметры и данные в качестве входных данных и возвращает оптимизированные параметры, которые лучше всего соответствуют данным. Вот пример использования функции curve_fit для подгонки синусоиды к набору точек данных:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

def sine_wave(x, a, b, c, d):
    return a * np.sin(b * x + c) + d

x_data = np.linspace(0, 10, 50)
y_data = sine_wave(x_data, 2, 0.5, 1, 0) + np.random.normal(0, 0.2, x_data.shape)

params, cov = curve_fit(sine_wave, x_data, y_data)
print(params)

plt.plot(x_data, y_data, 'o', label='data')
plt.plot(x_data, sine_wave(x_data, params[0], params[1], params[2], params[3]), '-',
label='fit')
plt.legend()
plt.show()
```

также предоставляет в scipy.optimizeмодуле ряд показателей и функций оптимизации. Например, функцию minimize можно использовать для минимизации заданной функции по отношению к ее входам, а функцию root_scalar можно использовать для нахождения корней заданной функции. Модуль metrics предоставляет ряд показателей

производительности, таких как среднеквадратическая ошибка, среднеквадратическая ошибка, средняя абсолютная ошибка и оценка R^2, которые можно использовать для оценки производительности модели.

Библиотека SciPy предоставляет ряд функций для оценки таких метрик, как среднеквадратическая ошибка, средняя абсолютная ошибка, оценка R^2 и другие. Вот пример использования среднеквадратичной ошибки:

```
import numpy as np
from scipy.optimize import curve_fit
from sklearn.metrics import mean_squared_error

# Generate data to fit
x_data = np.linspace(0, 10, 50)
y_data = 2 * x_data + 3 + np.random.normal(0, 1, x_data.shape)

# Define the curve to fit
def linear_function(x, a, b):
    return a * x + b

# Fit the curve to the data
params, cov = curve_fit(linear_function, x_data, y_data)
a, b = params

# Calculate the mean squared error
y_pred = linear_function(x_data, a, b)
mse = mean_squared_error(y_data, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
```

Линейная алгебра — это раздел математики, который занимается векторными пространствами и линейными преобразованиями между ними. Это фундаментальный инструмент во многих областях математики и естественных наук, включая физику, инженерию, информатику и экономику.

В Python самой популярной библиотекой для линейной алгебры является NumPy, которая предоставляет широкий набор функций и классов для работы с массивами и матрицами. Это также основа для других важных библиотек, таких как scikit-learn и pandas.

NumPy предоставляет функции для базовых операций линейной алгебры, таких как умножение матриц, инверсия и вычисление определителя, а также для более сложных операций, таких как разложение собственных и сингулярных значений.

Например, вы можете использовать функцию точки для выполнения матричного умножения:

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.dot(A, B)
print(C)
```

Вывод будет:

```
[[19 22]
 [43 50]]
```

Вы также можете использовать функцию inv для вычисления обратной матрицы:

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
A_inv = np.linalg.inv(A)
print(A_inv)
```

Вывод будет:

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Кроме того, модуль numpy.linalg также предоставляет функции для решения линейных систем уравнений, вычисления собственных значений и собственных векторов и разложения по сингулярным числам.

Линейная алгебра является фундаментальной концепцией в машинном обучении и науке о данных. Она используется во многих алгоритмах, таких как анализ главных компонентов (PCA), разложение по сингулярным значениям (SVD), линейный дискриминантный анализ (LDA) и многие другие.

Scikit-learn, библиотека машинного обучения, построенная на основе NumPy, использует возможности линейной алгебры NumPy и предоставляет широкий спектр моделей машинного обучения, которые можно использовать для таких задач, как классификация, регрессия и кластеризация.

Машинное обучение — это область искусственного интеллекта (ИИ), которая включает разработку алгоритмов и статистических моделей, которые позволяют компьютерам учиться и делать прогнозы или решения без явного программирования для этого.

Существует несколько видов машинного обучения:

- Контролируемое обучение: алгоритм обучается на размеченном наборе данных, где для каждого входа предоставляется правильный результат. Цель состоит в том, чтобы делать прогнозы на основе новых, невидимых данных. Примеры включают линейную регрессию, логистическую регрессию и машины опорных векторов.

- Неконтролируемое обучение: алгоритм обучается на немаркированном наборе данных, и цель состоит в том, чтобы обнаружить закономерности или структуру в данных. Примеры включают кластеризацию k-средних и анализ основных компонентов (PCA).

- Обучение с подкреплением: алгоритм учится, взаимодействуя с окружающей средой, получая награды или штрафы за определенные действия. Цель состоит в том, чтобы научиться принимать решения, которые со временем максимизируют совокупное вознаграждение.
- Глубокое обучение: Подобласть машинного обучения, которая включает обучение глубоких нейронных сетей (ГНС) на больших объемах данных. Было показано, что DNN достигают самых современных результатов в широком спектре задач, таких как классификация изображений, обработка естественного языка и распознавание речи.

Scikit-learn — это широко используемая библиотека для машинного обучения на Python. Она предоставляет широкий спектр инструментов и функций для контролируемого и неконтролируемого обучения, включая поддержку сред глубокого обучения, таких как TensorFlow и PyTorch.

Например, вы можете использовать scikit-learn для обучения модели линейной регрессии на наборе данных:

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston

# Load the Boston housing dataset
boston = load_boston()
X = boston.data
y = boston.target

# Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Make predictions
predictions = model.predict(X)
```

В этом примере класс LinearRegression используется для создания модели линейной регрессии, метод fit используется для обучения модели на наборе данных о жилье в Бостоне, а метод predict используется для прогнозирования новых данных.

Машинное обучение широко используется в NLP для таких задач, как:

- Классификация текста: классификация текста по предопределенным категориям, таким как обнаружение спама, анализ настроений и классификация тем.
- Распознавание именованных сущностей (NER): идентификация и извлечение сущностей, таких как люди, организации и местоположения, из текста.
- Тегирование части речи (POS): маркировка слов в предложении их грамматическими ролями, таких как существительные, глаголы и прилагательные.
- Моделирование языка: предсказание следующего слова в предложении по предыдущим словам.
- Машинный перевод: перевод текста с одного языка на другой.
- Обобщение текста: автоматическое обобщение основных моментов текста.
- Анализ тональности: определение тональности заданного текста.

Например, вы можете использовать библиотеку scikit-learn для обучения модели классификации текста в наборе данных:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

#define dataset
X_train = ["This is a positive text", "This is a negative text", "This is a neutral text"]
y_train = ["positive", "negative", "neutral"]
X_test = ["This is a positive text", "This is a negative text"]

#vectorize text
vectorizer = CountVectorizer()
X_train_vectors = vectorizer.fit_transform(X_train)
X_test_vectors = vectorizer.transform(X_test)

#train model
clf = MultinomialNB()
clf.fit(X_train_vectors, y_train)

#predict
predicted = clf.predict(X_test_vectors)
print("Accuracy: ", accuracy_score(predicted, y_test))
```

В этом примере набор данных представляет собой список текстов и соответствующих им меток («положительно», «отрицательно», «нейтрально»). Класс CountVectorizer используется для преобразования текстов в векторы числовых признаков, класс MultinomialNB используется для обучения наивной байесовской модели на наборе данных, а метод predict используется для прогнозирования новых данных.

Ещё один пример, в котором вы можете использовать библиотеку scikit-learn: обучение модели распознавания именованных объектов (NER) в наборе данных:

```
import nltk
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import DictVectorizer
from sklearn.tree import DecisionTreeClassifier

#define dataset
sentences = [["I", "live", "in", "New York"], ["I", "work", "at", "Google"]]
pos_tags = [["PRP", "VBP", "IN", "NNP"], ["PRP", "VB", "IN", "NNP"]]
ner_tags = [["O", "O", "O", "B-GPE"], ["O", "O", "O", "B-ORG"]]

#flatten the data
sentences = [token for sent in sentences for token in sent]
pos_tags = [tag for sent in pos_tags for tag in sent]
ner_tags = [tag for sent in ner_tags for tag in sent]

#extract features
def extract_features(sentence, index):
    features = {
        'word': sentence[index],
        'is_first': index == 0,
        'is_last': index == len(sentence) - 1,
        'is_capitalized': sentence[index][0].upper() == sentence[index][0],
        'is_all_caps': sentence[index].upper() == sentence[index],
        'is_all_lower': sentence[index].lower() == sentence[index],
        'prefix-1': sentence[index][0],
        'prefix-2': sentence[index][:2],
        'prefix-3': sentence[index][:3],
        'suffix-1': sentence[index][-1],
        'suffix-2': sentence[index][-2:],
        'suffix-3': sentence[index][-3:],
        'prev_word': '' if index == 0 else sentence[index - 1],
        'next_word': '' if index == len(sentence) - 1 else sentence[index + 1],
```

```
        'has_hyphen': '-' in sentence[index],
        'is_numeric': sentence[index].isdigit(),
        'capitals_inside': sentence[index][1:].lower() != sentence[index][1:]
    }
    return features

#extract features for each token
X = [extract_features(sentences, i) for i in range(len(sentences))]

#Vectorize the features
vec = DictVectorizer()
X = vec.fit_transform(X)

#split dataset into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, ner_tags, test_size=0.33)

#train the model
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

#predict
predicted = clf.predict(X_test)
print("Accuracy: ", accuracy_score(predicted, y)
```

Помимо классификации текста, еще одна распространенная задача в NLP, к которой можно подойти с помощью машинного обучения, называется распознаванием сущностей (NER). Это включает в себя идентификацию и классификацию именованных объектов, таких как люди, организации и местоположения, в тексте. Один из подходов к NER заключается в использовании алгоритма маркировки последовательностей, такого как скрытая марковская модель (HMM) или условное случайное поле (CRF), для прогнозирования меток для каждого слова в предложении. Другой подход заключается в использовании модели машинного обучения, такой как машина опорных векторов (SVM) или нейронная сеть, для классификации слов или фраз в тексте.

На практике многие задачи NLP, такие как определение частей речи и синтаксический анализ, также могут решаться с помощью машинного обучения. Еще одним примером задачи НЛП, к которой можно подойти с помощью машинного обучения, является анализ настроений. Анализ тональности — это процесс определения того, выражает ли фрагмент текста положительное, отрицательное или нейтральное настроение. Анализ настроений используется в широком спектре приложений, таких как обслуживание клиентов, маркетинг и анализ общественного мнения. Общие методы анализа настроений включают использование контролируемых алгоритмов машинного

обучения для классификации текста на основе наличия определенных ключевых слов или использование неконтролируемых методов для выявления шаблонов в тексте.

Лекция 5

Обрабюотка изображений

Обработка изображений — это метод обработки и анализа цифровых изображений, который часто используется в компьютерном зрении и компьютерной графике. OpenCV (Библиотека компьютерного зрения с открытым исходным кодом) — популярная библиотека с открытым исходным кодом для обработки изображений и компьютерного зрения на Python. Он предоставляет широкий спектр инструментов и алгоритмов для обработки изображений, таких как фильтрация изображений, пороговое значение, преобразование цветового пространства, обнаружение признаков и многое другое.

Одной из наиболее распространенных задач при обработке изображений является определение порога изображения, то есть процесс преобразования изображения в бинарное изображение (черно-белое) на основе порогового значения. OpenCV предоставляет несколько методов определения порога, таких как бинарное определение порога, адаптивное определение порога и определение порога Оцу.

Другой распространенной задачей обработки изображений является фильтрация изображений, то есть процесс изменения пикселей изображения на основе ядра фильтра. OpenCV предоставляет несколько методов фильтрации изображений, таких как размытие по Гауссу, срединное размытие и двусторонний фильтр.

Ещё одной распространенной задачей обработки изображений является обнаружение признаков, т. е. процесс обнаружения и извлечения элементов изображения, таких как углы, края и круги. OpenCV предоставляет несколько методов обнаружения функций, таких как обнаружение углов Харриса, обнаружение углов FAST и преобразование круга Хафа.

OpenCV может быть использовано для обнаружения лиц на изображении. Это можно сделать с помощью предварительно обученного классификатора, такого как алгоритм Виолы-Джонса, который включен в OpenCV. Другой задачей может быть использование OpenCV для обнаружения и отслеживания объектов в видеопотоке, что может быть выполнено с использованием различных алгоритмов, таких как вычитание фона, оптический поток и отслеживание объектов.

Примеры использования OpenCV для задач обработки изображений.

Пороговое значение: следующий код демонстрирует, как использовать OpenCV для выполнения двоичного порогового значения для изображения. Функция cv2.threshold() используется для преобразования изображения в двоичное изображение на основе порогового значения.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Perform binary thresholding
ret, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Show the original and thresholded images
cv2.imshow("Original", img)
cv2.imshow("Thresholded", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Фильтрация: следующий код демонстрирует, как использовать OpenCV для выполнения медианной фильтрации изображения. Функция cv2.medianBlur()используется для применения медианного фильтра к изображению.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Perform median filtering
median = cv2.medianBlur(img, 5)

# Show the original and filtered images
cv2.imshow("Original", img)
cv2.imshow("Median Filtered", median)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

Обнаружение функций. В следующем коде показано, как использовать OpenCV для обнаружения и рисования кругов на изображении. Эта cv2.HoughCircles()функция используется для обнаружения кругов на изображении с помощью преобразования круга Хафа.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect circles in the image
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 20, param1=50, param2=30, minRadius=0, maxRadius=0)

# Draw the circles on the image
for circle in circles[0, :]:
    cv2.circle(img, (circle[0], circle[1]), circle[2], (0, 255, 0), 2)

# Show the original and processed images
cv2.imshow("Original", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Бинаризация — это процесс преобразования изображения в бинарное изображение, где пиксели либо черные, либо белые. Порог для бинаризации обычно устанавливается на основе интенсивности или цвета изображения. Например, в изображениях в градациях серого пиксели со значениями интенсивности выше определенного порога устанавливаются белыми, а пиксели со значениями интенсивности ниже порога — черными. В OpenCV доступно несколько методов бинаризации, таких как бинаризация Оцу, адаптивная бинаризация и бинаризация Ниблэка.

Бинаризация особенно полезна в приложениях, где изображения используются для распознавания символов, например, при оптическом распознавании символов (OCR) и распознавании рукописного ввода. Например, в OCR целью является точное распознавание текста на изображениях отсканированных документов, а для отделения текста от фона используется бинаризация. Точно так же при распознавании рукописного ввода бинаризация используется для отделения письменных символов от бумаги.

Распознавание автомобильных номеров — еще одно приложение, использующее бинаризацию. В этом приложении цель состоит в том, чтобы точно распознать символы на номерном знаке автомобиля. Бинаризация используется для отделения символов от фона, что упрощает идентификацию символов алгоритму распознавания.

Существуют различные методы бинаризации, в том числе глобальное пороговое значение, адаптивное пороговое значение и метод Оцу. Глобальное пороговое значение задает одно пороговое значение для всего изображения, а пиксели выше порога устанавливаются белыми, а пиксели ниже порога — черными. Адаптивная пороговая установка устанавливает пороговое значение для каждого отдельного пикселя на основе значений окружающих пикселей. Метод Оцу автоматически определяет оптимальное пороговое значение на основе гистограммы изображения.

Дилатация и эрозия — это операции морфологической обработки изображения, которые используются для расширения или сжатия границ объекта на изображении. Расширение увеличивает размер объекта, добавляя пиксели к границе объекта, а эрозия уменьшает размер объекта, удаляя пиксели с границы объекта. Эти операции полезны для удаления шума и небольших отверстий в изображении.

Методы поиска контуров на изображении включают использование алгоритмов обнаружения краев, таких как детектор краев Кэнни, или использование методов пороговой обработки. Контуры можно использовать для идентификации и сегментации объектов на изображении.

Обнаружение и сегментация объектов на изображении может быть достигнуто с помощью нескольких методов, таких как определение порога, обнаружение краев и обнаружение контуров. Эти методы можно использовать для идентификации и выделения определенных объектов на изображении, например объектов определенного типа или определенного цвета.

Например, используя OpenCV, мы можем загрузить изображение, а затем использовать детектор краев Канни, чтобы найти края на изображении. Затем мы можем использовать метод findContours, чтобы найти контуры на изображении. Затем мы можем нарисовать контуры на изображении, чтобы сегментировать объекты на изображении.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Use Canny edge detector to find edges in the image
edges = cv2.Canny(gray, 50, 150)

# Find contours in the image
contours, hierarchy = cv2.findContours(edges, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Draw the contours on the image
cv2.drawContours(img, contours, -1, (0, 255, 0), 2)

# Show the image
cv2.imshow("Segmented Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вы также можете использовать inRangeметод для порогового изображения и сегментации объекта определенного цвета.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")
```

```
# Define the range of the color you want to segment
lower = (0, 0, 0)
upper = (50, 50, 50)

# Threshold the image to get only the desired color
mask = cv2.inRange(img, lower, upper)

# Show the segmented image
cv2.imshow("Segmented Image", mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Лекция 6

Парсинг сайтов

Построение простых запросов с помощью библиотеки Request. GET и POST запросы. Парсинг сайтов LXML. ETree. BeautifulSoup. Обработка текстовой информации, полученной с сайтов.

Веб-скрапинг — это процесс автоматического извлечения информации с веб-сайтов, часто больших объемов данных, с целью преобразования ее в структурированный формат для анализа или хранения. Цель парсинга веб-страниц — извлечение данных с веб-сайтов, как правило, путем имитации просмотра человеком веб-страниц, а также извлечение конкретной информации, которая может использоваться для различных целей, таких как анализ данных, машинное обучение и т. д.

Разница между скрапингом и парсингом (синтаксическим анализом) заключается в том, что парсинг обычно относится к процессу разбиения большого фрагмента данных на более мелкие, более управляемые части. В контексте скрапинга веб-страниц синтаксический анализ относится к процессу извлечения определенной информации из HTML-ответа, полученного с веб-сайта. Скрапинг, с другой стороны, включает в себя весь процесс отправки HTTP-запроса, получения ответа HTML и последующего анализа ответа для извлечения нужной информации.

requests — это библиотека для создания HTTP-запросов, которую можно использовать для широкого круга приложений, включая синтаксический анализ сайтов.

Простой пример использования библиотеки requests для выполнения GET-запроса к веб-сайту:

```
import requests

url = "https://www.example.com"
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    # Get the content of the response
    content = response.content
    print(content)
```

```
else:
    # Print the error message
    print("Request failed with status code:", response.status_code)
```

В этом примере функция requests.get используется для отправки запроса GET на указанный URL-адрес. Объект response, возвращаемый функцией, содержит информацию об ответе, включая код состояния и содержимое. Код состояния проверяется, чтобы убедиться, что запрос был успешным, и, если это так, выводится содержимое ответа.

Аналогичный подход можно использовать для выполнения POST-запросов с помощью функции requests.post. Функция принимает URL-адрес, данные для отправки и любые заголовки в качестве аргументов.

```
import requests

url = "https://www.example.com/submit"
data = {
    "field1": "value1",
    "field2": "value2",
}
headers = {
    "Content-Type": "application/x-www-form-urlencoded",
}

response = requests.post(url, data=data, headers=headers)

# Check if the request was successful
if response.status_code == 200:
    # Get the content of the response
    content = response.content
    print(content)
else:
    # Print the error message
    print("Request failed with status code:", response.status_code)
```

В этом примере POST-запрос отправляется на URL-адрес с указанными данными и заголовками. Код состояния ответа проверяется, и содержимое ответа печатается, если запрос был успешным.

LXML — это мощная библиотека для обработки XML- и HTML-документов, что делает ее популярным выбором для парсинга веб-страниц.
В этой лекции мы узнаем об основах LXML и о том, как использовать его для извлечения информации с веб-сайтов. Мы начнем с обсуждения базовой структуры документа XML или HTML и того, как его анализировать с помощью LXML.
Далее мы узнаем о различных функциях и методах, доступных в LXML для выбора и извлечения элементов с веб-сайта. Мы также обсудим различные способы обработки данных после их извлечения, например преобразование их в более удобный формат или сохранение в базе данных.
Наконец, мы узнаем, как использовать LXML в сочетании с другими библиотеками Python, такими как Requests, для создания более сложных проектов парсинга веб-страниц.
К концу этой лекции вы будете хорошо понимать, как использовать LXML для анализа веб-сайтов и извлечения информации из них. Обладая этими знаниями, вы сможете создавать свои собственные веб-проекты и автоматизировать задачи, которые в противном случае потребовали бы ручного труда.

Чтобы начать работу с LXML, вам необходимо установить его, запустив pip install lxml. После установки вы можете импортировать библиотеку в свой скрипт Python, запустив import lxml.

Для анализа веб-сайтв с помощью LXML, вам сначала необходимо получить данные HTML или XML с сайта. Вы можете сделать это с помощью requestsбиблиотеки Python, которая предоставляет простой способ выполнения HTTP-запросов. Например, следующий код извлечет HTML-код с веб-сайта и сохранит его в переменной:

```
import requests

response = requests.get("https://example.com")
html = response.text
```

Когда у вас есть данные HTML, вы можете проанализировать их, используя метод LXML fromstring. Этот метод принимает строку данных HTML или XML и возвращает объект, которым вы можете манипулировать. Например:

```
from lxml import html
root = html.fromstring(html)
```

Результирующий rootобъект представляет собой корневой узел документа HTML или XML. Затем вы можете использовать методы, предоставляемые LXML, для извлечения информации из документа. Например, вы можете использовать этот xpathметод для поиска элементов с определенными именами тегов или атрибутами:

```
headings = root.xpath("//h1")
```

Этот код вернет список всех <h1>элементов HTML-документа. Затем вы можете перебрать список и извлечь текстовое содержимое каждого заголовка:

```
for heading in headings:
    print(heading.text)
```

LXML также предоставляет методы для управления документом, такие как добавление или удаление элементов, изменение значений атрибутов или добавление текстового содержимого. Вы можете использовать эти методы для изменения данных HTML или XML, а затем сохранить их в файл или отправить обратно на сервер.

В LXML доступно гораздо больше функций и методов, поэтому обязательно ознакомьтесь с официальной документацией для получения дополнительной информации: https://lxml.de/

lxml.etree используется для работы с документами XML и предоставляет надежный и многофункциональный API для анализа, обработки и создания XML и HTML. Он основан на ElementTree API, который включен в стандартную библиотеку Python, но добавляет множество дополнительных функций и оптимизаций.

lxml.html используется для работы с документами HTML и предоставляет удобный API для анализа и управления HTML, а также очистки HTML, чтобы сделать его правильно сформированным и готовым к обработке.

И lxml.etree, и lxml.html работают быстро и эффективно используют память, что делает их подходящими для работы с большими документами или очистки большого количества веб-страниц. Они также поддерживают широкий спектр типов кодировок, включая Unicode и различные кодировки ISO-8859, что делает их подходящими для работы с документами на нескольких языках.

ElementTree является частью стандартной библиотеки Python и используется для работы с файлами XML и HTML. Он предоставляет простой способ анализа файлов XML/HTML и управления ими.

Основным классом в ElementTree является класс Element, представляющий XML-элемент. Он имеет методы для доступа и изменения тега элемента, атрибутов, текста и дочерних элементов.

Вот базовый пример того, как использовать ElementTree для разбора XML-файла и доступа к его элементам:

```
import xml.etree.ElementTree as ET

# parse an XML file
tree = ET.parse('sample.xml')
root = tree.getroot()

# access elements
for child in root:
    print(child.tag, child.attrib)
    for subchild in child:
        print(subchild.tag, subchild.attrib)

# modify elements
for elem in root.iter('value'):
    elem.text = 'new value'

# write the modified XML to a file
tree.write('modified.xml')
```

В этом примере показано, как проанализировать файл XML с помощью ElementTree, получить доступ к его элементам, изменить их значения и записать измененный XML обратно в файл.
После того, как вы проанализировали содержимое в структуру ElementTree, вы можете использовать различные методы для извлечения интересующей вас информации. Например, вы можете использовать метод xpath для выбора элементов на основе их имен тегов, классов или других атрибутов. , или вы можете использовать метод cssselect для выбора элементов на основе их селекторов CSS.
В дополнение к этим базовым возможностям синтаксического анализа и извлечения LXML также предоставляет ряд дополнительных функций и инструментов, включая поддержку синтаксического анализа и проверки документов на соответствие DTD или XML-схемам, поддержку преобразования

XML-документов с использованием таблиц стилей XSLT и поддержку работы с HTML-формами. и отправка данных на веб-сайты.

Пример анализа HTML-документа с помощью LXML и извлечения информации:

```
import lxml.html

# Load the HTML document into memory
html = lxml.html.fromstring("<html><body><h1>Hello World!</h1></body></html>")

# Use the xpath method to select the h1 element
h1 = html.xpath("//h1")[0]

# Extract the text content of the h1 element
print(h1.text)  # "Hello World!"
```

Пример анализа XML-документа с помощью LXML и его преобразования с использованием таблицы стилей XSLT:

```
import lxml.etree

# Load the XML document into memory
xml = lxml.etree.fromstring("<data><record><name>John Doe</name><age>30</age></record></data>")

# Load the XSLT stylesheet into memory
xslt = lxml.etree.fromstring("""
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <html>
      <body>
        <table>
          <tr>
            <th>Name</th>
            <th>Age</th>
          </tr>
          <xsl:for-each select="data/record">
            <tr>
              <td><xsl:value-of select="name" /></td>
```

```
            <td><xsl:value-of select="age" /></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
""")

# Transform the XML document using the XSLT stylesheet
result = lxml.etree.XSLT(xslt)(xml)

# Print the transformed HTML document
print(lxml.etree.tostring(result, pretty_print=True).decode())
```

BeautifulSoup — еще одна популярная библиотека для веб-скрапинга и анализа документов HTML и XML. Как и LXML, он предоставляет способ извлечения информации с веб-сайтов, но имеет немного другую направленность и подход. В то время как LXML является более полной библиотекой, которая предоставляет широкий спектр функций для манипулирования и преобразования документов XML и HTML, BeautifulSoup более конкретно ориентирован на задачу очистки веб-страниц и предоставляет более простой и понятный интерфейс для извлечения информации с веб-сайтов.

BeautifulSoup особенно хорошо подходит для таких задач, как извлечение текстового содержимого из HTML-страниц, извлечение данных из таблиц или поиск определенных элементов на странице на основе имен их тегов, классов или атрибутов. Чтобы использовать BeautifulSoup, вам сначала нужно установить библиотеку и импортировать ее в свой код Python, а затем использовать функцию BeautifulSoup для анализа документа HTML или XML, который вы хотите очистить.

После того, как вы проанализировали документ, вы можете использовать различные методы для поиска и извлечения интересующей вас информации. Например, вы можете использовать методы find и find_all для поиска определенных элементов на странице, или вы можете использовать метод select для поиска элементов, соответствующих определенному селектору CSS.

Одной из ключевых особенностей BeautifulSoup является возможность поиска тегов и элементов с использованием имен тегов, атрибутов класса и идентификатора, а также селекторов CSS. Например, вы можете использовать метод «выбрать», чтобы выбрать элементы на основе их селекторов CSS, или метод «найти», чтобы найти первый элемент, соответствующий заданному имени тега или атрибуту.

Важной особенностью BeautifulSoup является его способность обрабатывать искаженный HTML-код. В отличие от других библиотек, которые могут дать сбой при синтаксическом анализе искаженного HTML, BeautifulSoup может анализировать и извлекать данные даже из самых плохо отформатированных HTML-документов.

Пример использования BeautifulSoup для извлечения данных из HTML-документа:

```
from bs4 import BeautifulSoup

html_doc = """
<html>
  <head>
    <title>Example Page</title>
  </head>
  <body>
    <h1>Welcome to my page</h1>
    <p>Here is some content</p>
    <p class="highlight">And here is a highlighted paragraph</p>
  </body>
</html>
"""

soup = BeautifulSoup(html_doc, "html.parser")

# Extract the title tag
title = soup.title

# Extract the first h1 tag
h1 = soup.h1

# Extract all p tags
p_tags = soup.find_all("p")
```

```
# Extract the first p tag with class "highlight"
highlighted_p = soup.select_one(".highlight")

print("Title:", title.text)
print("H1:", h1.text)
print("P tags:")
for p in p_tags:
    print(p.text)
print("Highlighted P:", highlighted_p.text)
```

Этот код выведет следующее:

```
Title: Example Page
H1: Welcome to my page
P tags:
Here is some content
And here is a highlighted paragraph
Highlighted P: And here is a highlighted paragraph
```

После того, как вы извлекли информацию с веб-сайта с помощью LXML или BeautifulSoup, следующим шагом будет обработка полученной текстовой информации. Это может включать в себя ряд задач, таких как очистка и предварительная обработка данных, преобразование их в формат, подходящий для анализа, или извлечение определенной информации из текста, такой как именованные объекты, ключевые слова или настроения.

Одним из распространенных примеров обработки текстовой информации с веб-сайта является извлечение именованных сущностей из новостной статьи. Например, рассмотрим следующую статью:

Microsoft has announced that it is acquiring LinkedIn for $26.2 billion. The deal is expected to close by the end of 2016.

Чтобы извлечь именованные объекты из этой статьи, вам сначала потребуется очистить и предварительно обработать текст, например преобразовать все символы в нижний регистр и удалить знаки препинания. Затем вы должны использовать инструмент распознавания именованных объектов (NER), например Распознаватель именованных объектов (NER) в наборе инструментов естественного языка (NLTK), для идентификации именованных объектов в тексте.

Вот пример того, как это можно сделать в Python с помощью NLTK:

```
import nltk
from nltk.tokenize import word_tokenize

text = "Microsoft has announced that it is acquiring LinkedIn for $26.2 billion. The deal is expected to close by the end of 2016."

# Tokenize the text
tokens = word_tokenize(text)

# Part-of-speech tag the tokens
tagged = nltk.pos_tag(tokens)

# Identify named entities
named_entities = nltk.ne_chunk(tagged)

# Print the named entities
for entity in named_entities:
    if hasattr(entity, 'label'):
        print(entity.label(), ' '.join(c[0] for c in entity.leaves()))
```

Этот код выдаст следующий результат:

```
ORGANIZATION Microsoft
ORGANIZATION LinkedIn
MONEY $26.2 billion
```

Как видите, инструмент NER правильно идентифицировал названные сущности «Microsoft» и «LinkedIn» как организации, а сумму «26,2 миллиарда долларов» — как деньги.

Лекция 7

Архитектура приложений. API

Архитектура приложения относится к дизайну базовых систем приложения, включая структуру его компонентов, отношения между этими компонентами и принципы, которыми руководствуется их дизайн. Хорошо спроектированная архитектура приложения может помочь обеспечить его масштабируемость, удобство обслуживания и простоту разработки и развертывания.

API или интерфейс прикладного программирования — это набор протоколов, подпрограмм и инструментов для создания программных приложений. Он определяет, как должны взаимодействовать программные компоненты, а API-интерфейсы обеспечивают связь между различными приложениями или системами. API также можно использовать для доступа к удаленным системам или данным, что упрощает разработчикам интеграцию их приложений с другими службами.

Обычно API-интерфейсы используются при создании микросервисов, когда приложение разбивается на более мелкие повторно используемые компоненты, которыми можно управлять и разрабатывать независимо. API также можно использовать для предоставления данных и функций внешним пользователям, что позволяет им создавать собственные приложения, взаимодействующие с системой.

Например, служба погоды может предоставить API, который позволяет разработчикам получать доступ к своим данным и использовать их в своих собственных приложениях. Другим примером является использование API-интерфейсов на платформах социальных сетей, которые позволяют разработчикам получать доступ к информации о пользователях, сообщениях и другим данным и использовать их для создания собственных приложений.

Следует отметить, что архитектура приложений и API-интерфейсы играют решающую роль в современной разработке программного обеспечения и необходимы для создания масштабируемых, гибких и удобных в сопровождении приложений.

Пример простого API, написанного на Python с использованием библиотеки Flask:

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/api', methods=['GET', 'POST'])
def handle_api_request():
    if request.method == 'GET':
        # handle GET request
        return 'Handling GET request'
    elif request.method == 'POST':
```

```
        # handle POST request
        return 'Handling POST request'

if __name__ == '__main__':
    app.run()
```

В этом примере настраивается базовый API с одной конечной точкой, которая обрабатывает запросы GET и POST. При получении запроса GET API вернет строку «Обработка запроса GET», а при получении запроса POST — строку «Обработка запроса POST».